机器学习第一次作业

1、公式推导

证明：$f_\theta(x_i) = \frac{1}{1+e^{-\theta^T x_i}}$ $\quad 1-f_\theta(x_i) = \frac{e^{-\theta^T x_i}}{1+e^{-\theta^T x_i}}$

$$\ln(L(\theta)) = \sum_{i=1}^{n} (y_i * \ln f_\theta(x_i) + (1-y_i) \ln (1-f_\theta(x_i))$$

$$= \sum_{i=1}^{n} (y_i (\ln f_\theta(x_i) - \ln(1-f_\theta(x_i))) + \ln(1-f_\theta(x_i)))$$

$$= \sum_{i=1}^{n} (y_i * \ln \frac{f_\theta(x_i)}{1-f_\theta(x_i)} + \ln(1-f_\theta(x_i)))$$

$$= \sum_{i=1}^{n} (y_i * \ln \frac{1}{e^{-\theta^T x_i}} + \ln(\frac{e^{-\theta^T x_i}}{1+e^{-\theta^T x_i}} \cdot \frac{e^{\theta^T x_i}}{e^{\theta^T x_i}}))$$

$$= \sum_{i=1}^{n} (y_i * \theta^T * x_i + \ln \frac{1}{e^{\theta^T x_i}+1})$$

$$= \sum_{i=1}^{n} (y_i * \theta^T * x_i - \ln(e^{\theta^T x_i}+1))$$

$$= \sum_{i=1}^{n} ((y_i * \theta^T * x_i) - \ln(1+e^{\theta^T x_i}))$$

∴原式成立

2、上机实验

（1）从文件中读取数据，直接转换为四阶多项式，以四阶多项式作为决策边界。

```python
def read_data(string):
    infile = open(string, 'r')
    data, l_x, l_y = [], [], []
    for line in infile:
        words = line.split(',')  # 以逗号分开
        x1 = float(words[0])
        x2 = float(words[1])
        y1 = int(words[2][0:1])
        l_x.append([1, x1, x2, x1 * x2, x1 ** 2, x2 ** 2,
                    x1 * x2 ** 2, x2 * x1 ** 2, x1 ** 3, x2 ** 3,
                    x1 * x2 ** 3, x1 ** 2 * x2 ** 2, x1 ** 3 * x2, x1 ** 4, x2 ** 4])
        l_y.append([y1])
        data.append([x1, x2, y1])
    infile.close()
    l_x = np.array(l_x)
    l_y = np.array(l_y)
    data = np.array(data)
    return data, l_x, l_y
```

（2）梯度下降，在 θ 或者损失函数下降很小时提前终止

```python
def gradient_ascent(X, y, eta=0.1, n_iterations=5000):
    theta = np.random.randn(15, 1)
    for iteration in range(n_iterations):
        f = sigmoid(np.dot(X, theta))
        gradients = np.dot(X.T, (y - f))
        theta_old = np.array([i for i in theta])
        theta += eta * gradients
        if 0.01 < cost(theta_old, X, y) - cost(theta, X, y) < 0.01:
            return theta
        if 0.01 < np.dot((theta - theta_old).T, (theta - theta_old)) < 0.01:
            return theta
    return theta
```

（3）随机梯度下降，随机选择一个样本进行迭代， 每轮迭代之后降低学习率，减小振荡

```python
def stochastic_gradient_ascent(X, y):
    theta = np.random.randn(15, 1)
    for epoch in range(100):
        for i in range(len(y)):
            random_index = np.random.randint(len(y))
            xi = X[random_index:random_index + 1, :]
            yi = y[random_index:random_index + 1]
            f = sigmoid(np.dot(xi, theta))
            gradients = np.dot(xi.T, (yi - f))
            eta = 1 / (np.sqrt(epoch + 1))  # 不断降低学习率
            theta += eta * gradients
    return theta
```

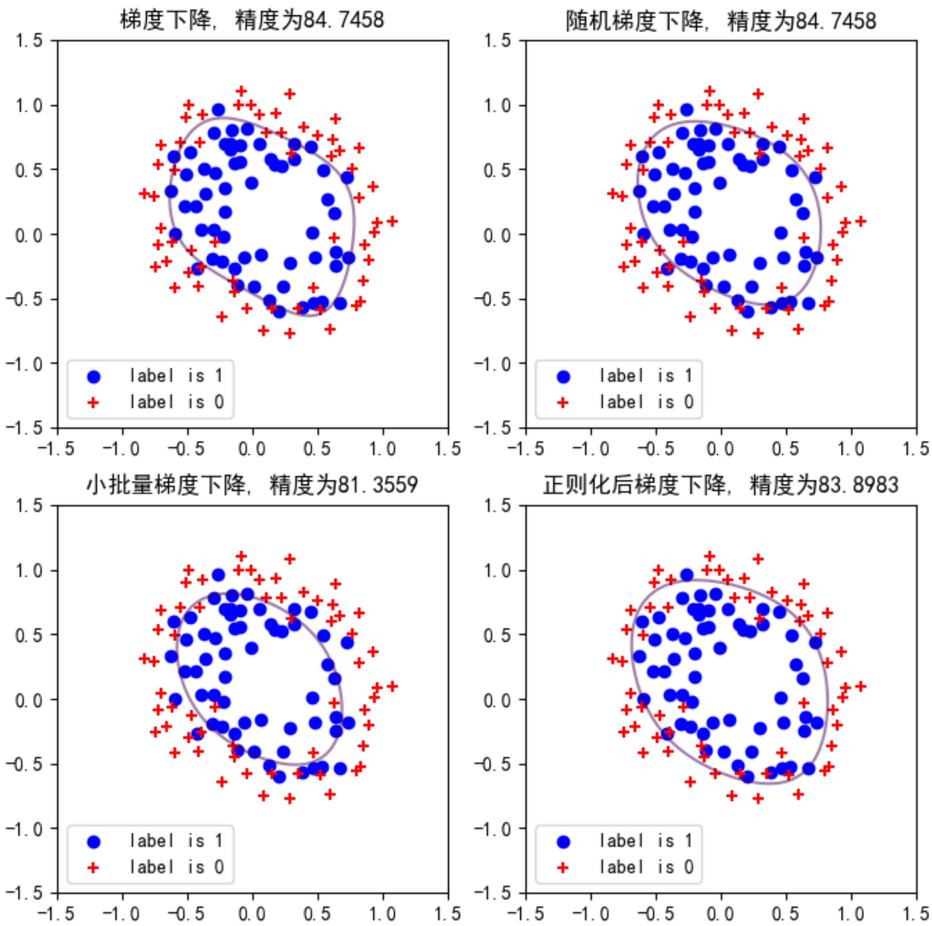
（4）小批量梯度下降，每次选择 mb_size 个样本，事先按顺序分配好迭代的小样本，然后进行 100 轮迭代。

```python
def mini_batch_gradient_ascent(X, y, mb_size=10, eta=0.1):
    theta = np.random.randn(15, 1)
    m = len(y)
    nums = (m - 1) // mb_size
    index_list = np.arange(nums)
    index_list = index_list * mb_size
    for epoch in range(100):
        for index in index_list:
            xi = X[index:index + mb_size + 1, :]
            yi = y[index:index + mb_size + 1]
            f = sigmoid(np.dot(xi, theta))
            gradients = np.dot(xi.T, (yi - f))
            theta += eta * gradients
        # 处理最后可能不到mb_size大小的数组
        d = m - nums * mb_size
        index = index_list[-1] + mb_size
        xi = X[index:index + d + 1, :]
        yi = y[index:index + d + 1]
        f = sigmoid(np.dot(xi, theta))
        gradients = np.dot(xi.T, (yi - f))
        theta += eta * gradients
    return theta
```

（5）正则化，对 θ 进行惩罚，减小过拟合，以梯度下降为例

```python
def reg_gradient(X, y, eta=0.1, n_iterations=10000, learning_rate=0.1):
    theta = np.random.randn(15, 1)
    for iteration in range(n_iterations):
        f = sigmoid(np.dot(X, theta))
        gradients = np.dot(X.T, (y - f))
        theta_old = np.array([i for i in theta])
        theta = theta * (1 - eta * learning_rate) + eta * gradients
        if 0.01 < reg_cost(theta_old, X, y) - reg_cost(theta, X, y) < 0.01:
            return theta
        if 0.01 < np.dot((theta - theta_old).T, (theta - theta_old)) < 0.01:
            return theta
    return theta
```

（6）梯度下降，随机梯度下降， 小批量梯度下降和正则化梯度下降进行比较，可以发现都可以得到较好的结果。加入正则化后梯度下降的 θ 值更小一点，决策边界更光滑些，较好的减小了过拟合的现象。



梯度下降后的得到的θ值为：

[4.15760623 1.58868227 4.00746323 -9.94403463 -5.22985051 -8.27080939
  7.67129608 0.38400842 2.34300899 2.60837205 -8.87190575 -8.7397435
 10.54764701 -9.03176705 -6.07167211]

随机梯度下降后的得到的θ值为：

[4.18732136 2.52144889 4.25928263 -6.03602321 -6.85404428 -7.17677913
 -0.69428872 -1.66590994 1.18860034 -1.57376916 -2.00956159 -2.19906501
  1.1130972 -6.16796198 -4.46086954]

小批量梯度下降后的得到的θ值为：

[3.05340103 2.02688606 3.52050902 -5.90566788 -7.87894932 -6.39295466
 -0.88827487 -1.98658334 0.56767954 0.31943746 -1.51214073 -3.08040269
 -0.19991322 -4.84840199 -5.22576131]

对梯度下降正则化后的得到的θ值为：

[2.37759695 1.85806816 3.08178778 -3.60275245 -5.01716638 -4.59436536
 -0.63772071 -1.44076813 0.25959965 -1.06368007 -1.42729913 -1.97427622
  0.7225335 -4.53518934 -3.75507604]

（5）对正则化惩罚因子的比较，可以发现在惩罚因子比较好时，得到的结果比较好，但是在惩罚因子过大时，会找不到决策边界。